(http://baeldung.com)

Entity To DTO Conversion for a Spring REST API

Last modified: July 19, 2017

by baeldung (http://www.baeldung.com/author/baeldung/)

REST (http://www.baeldung.com/category/rest/) Spring (http://www.baeldung.com/category/spring/) +

1. Overview

In this tutorial, we'll handle the conversions that need to happen **between the internal entities of a Spring application and the external DTOs** (Data Transfer Objects) that are published back to the client.

Further reading:

A Guide to Mapping With Dozer (http://www.baeldung.com/dozer)

Dozer is a Java Bean to Java Bean mapper that copies data from one object to another, attribute by attribute, supports mapping between attribute names, does type conversion, and many other things.

Read more (http://www.baeldung.com/dozer) →

Quick Guide to MapStruct (http://www.baeldung.com/mapstru

A quick and practical guide to using MapStruct

Read more (http://www.baeldung.com/mapstruct) →

2. Model Mapper

Let's start by introducing the main library that we're going to use to perform this entity-DTO conversion – *ModelMapper (http://modelmapper.org/getting-started/)*.

We will need this dependency in the *pom.xml*:

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>0.7.4</version>
</dependency>
```

To check if there's any newer version of this library, go here (http://search.maven.org/#search|gav|1|g%3A%22org.modelmapper%22%20AND%20a%3A%22modelmapper%22).

We'll then define the *ModelMapper* bean in our Spring configuration:

```
@Bean
public ModelMapper modelMapper() {
    return new ModelMapper();
}
```

3. The DTO

Next, let's introduce the DTO side of this two-sided problem – *Post* DTO:

```
public class PostDto {
    private static final SimpleDateFormat dateFormat
      = new SimpleDateFormat("yyyy-MM-dd HH:mm");

    private Long id;

    private String title;

    private String url;

    private String date;

    private UserDto user;

    public Date getSubmissionDateConverted(String timezone) throws ParseException {
        dateFormat.setTimeZone(TimeZone.getTimeZone(timezone));
        return dateFormat.parse(this.date);
    }

    public void setSubmissionDate(Date date, String timezone) {
        dateFormat.setTimeZone(TimeZone.getTimeZone(timezone));
        this.date = dateFormat.format(date);
    }

    // standard getters and setters
}
```

Note that the 2 custom date related methods handle the date conversion back and forth between the client and the server:

- *getSubmissionDateConverted()* method converts Date String into a date in server's timezone to use it in persisting Post entity
- *setSubmissionDate()* method is to set DTO's date to Post's Date in current user timezone.

4. The Service Layer

Let's now look at a service level operation – which will obviously work with the Entity (not the DTO):

```
public List<Post> getPostsList(
  int page, int size, String sortDir, String sort) {

    PageRequest pageReq
     = new PageRequest(page, size, Sort.Direction.fromString(sortDir), sort);

    Page<Post> posts = postRepository
      .findByUser(userService.getCurrentUser(), pageReq);
    return posts.getContent();
}
```

We're going to have a look at the layer above service next – the controller layer. This is where the conversion will actually happen as well.

5. The Controller Layer

Let's now have a look at a standard controller implementation, exposing the simple REST API for the *Post* resource.

We're going to show here a few simple CRUD operations: create, update, get one and get all. And given the operations are pretty straightforward, **we are especially interested in the Entity-DTO conversion aspects**:

```
@Controller
class PostRestController {

    @Autowired
    private IPostService postService;

    @Autowired
    private IUserService userService;

    @Autowired
    private ModelMapper modelMapper;

    @RequestMapping(method = RequestMethod.GET)
    @ResponseBody
    public List<PostDto> getPosts(...) {
        //...
        List<Post> posts = postService.getPostsList(page, size, sortDir, sort);
        return posts.stream()
          .map(post -> convertToDto(post))
          .collect(Collectors.toList());
    }

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    @ResponseBody
    public PostDto createPost(@RequestBody PostDto postDto) {
        Post post = convertToEntity(postDto);
        Post postCreated = postService.createPost(post));
        return convertToDto(postCreated);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseBody
    public PostDto getPost(@PathVariable("id") Long id) {
        return convertToDto(postService.getPostById(id));
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void updatePost(@RequestBody PostDto postDto) {
        Post post = convertToEntity(postDto);
        postService.updatePost(post);
    }
}
```

And here is our conversion **from *Post* entity to *PostDto***:

```
private PostDto convertToDto(Post post) {
    PostDto postDto = modelMapper.map(post, PostDto.class);
    postDto.setSubmissionDate(post.getSubmissionDate(),
        userService.getCurrentUser().getPreference().getTimezone());
    return postDto;
}
```

And here is the conversion **from DTO to an entity**:

```
private Post convertToEntity(PostDto postDto) throws ParseException {
    Post post = modelMapper.map(postDto, Post.class);
    post.setSubmissionDate(postDto.getSubmissionDateConverted(
      userService.getCurrentUser().getPreference().getTimezone()));

    if (postDto.getId() != null) {
        Post oldPost = postService.getPostById(postDto.getId());
        post.setRedditID(oldPost.getRedditID());
        post.setSent(oldPost.isSent());
    }
    return post;
}
```

So, as you can see, with the help of the model mapper, **the conversion logic is quick and simple** – we're using the *map* API of the mapper and getting the data converted without writing a single line of conversion logic.

6. Unit Testing

Finally, let's do a very simple test to make sure the conversions between the entity and the DTO work well:

```
public class PostDtoUnitTest {

    private ModelMapper modelMapper = new ModelMapper();

    @Test
    public void whenConvertPostEntityToPostDto_thenCorrect() {
        Post post = new Post();
        post.setId(Long.valueOf(1));
        post.setTitle(randomAlphabetic(6));
        post.setUrl("www.test.com");

        PostDto postDto = modelMapper.map(post, PostDto.class);
        assertEquals(post.getId(), postDto.getId());
        assertEquals(post.getTitle(), postDto.getTitle());
        assertEquals(post.getUrl(), postDto.getUrl());
    }

    @Test
    public void whenConvertPostDtoToPostEntity_thenCorrect() {
        PostDto postDto = new PostDto();
        postDto.setId(Long.valueOf(1));
        postDto.setTitle(randomAlphabetic(6));
        postDto.setUrl("www.test.com");

        Post post = modelMapper.map(postDto, Post.class);
        assertEquals(postDto.getId(), post.getId());
        assertEquals(postDto.getTitle(), post.getTitle());
        assertEquals(postDto.getUrl(), post.getUrl());
    }
}
```

7. Conclusion

This was an article on **simplifying the conversion from Entity to DTO and from DTO to Entity in a Spring REST API**, by using the model mapper library instead of writing these conversions by hand.